Lecture 7

Ben Rosenberg

June 16, 2021

Review: CFGs

$\textcircled{1}$ Consider $\Sigma = \{\texttt{a},\texttt{b},\texttt{c}\}$, and a grammar for the set of strings whose length is divisible by 3. (Lengths are of the form $3n$ for some natural number n.)

Regular expression:

$$\Big((\texttt{a} \cup \texttt{b} \cup \texttt{c})(\texttt{a} \cup \texttt{b} \cup \texttt{c})(\texttt{a} \cup \texttt{b} \cup \texttt{c})\Big)^*$$

Grammar:

- $G = (V, \Sigma, P, S)$
- $V = \{S, Z\}$
- $P : S \to \lambda|SS|ZZZ;\ Z \to \texttt{a}|\texttt{b}|\texttt{c}$

Consider now grammar $G_1 = (V, \Sigma, P, S)$.

V:

- $S \implies$: strings with length $= 3n$.
- $N \implies$: strings with length $= 3n + 1$
- $T \implies$: strings with length $= 3n + 2$
- $Z \implies$: any single letter

P:

- $S \to \lambda|ZT$
- $T \to ZN$
- $N \to ZS$
- $Z \to \texttt{a}|\texttt{b}|\texttt{c}$

This grammar is equivalent to the previous grammar G, and while less intuitive, it has some advantages. For instance, if we wanted to get strings of length $3n + 1$, we would just make N the start symbol instead of S.

Finite state automata

A finite state machine has a box, that cannot be looked into. It has a dial, and a hand that moves on the dial, with indicatory marks that may be labeled. We call this the *state box*.

We then have an input of a string, which is unbounded but finite. The input is an element of Σ^*. There is a head on the tape that can only read the tape and move to the next symbol (to the right) after reading.

Initially, the state is some initial state q_0, and the tape will be pointing at the leftmost symbol of the input. For each symbol the machine reads, it looks at the symbol under the head, looks at the current state, consults its own definition, and changes its state (after which the head progresses one symbol to the right).

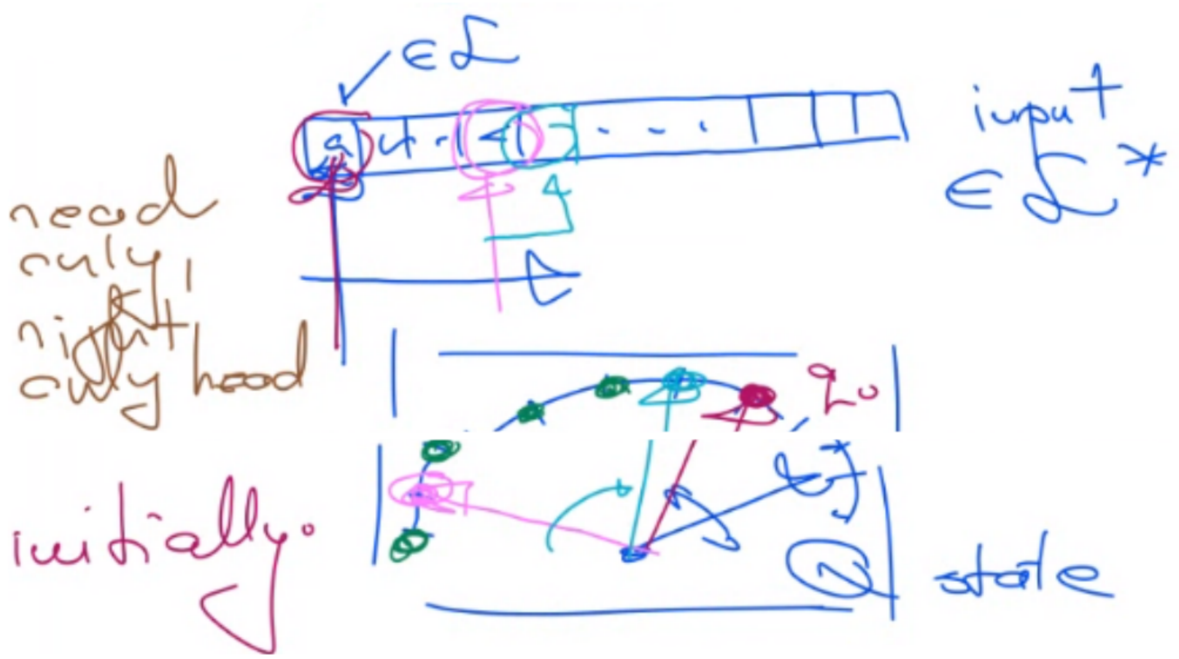

Figure 1: Illustration of analogy to finite state automaton

DFAs

Definition: A **deterministic finite state automaton** (DFA) is a structure $M = (Q, \Sigma, \delta, q_0, F)$.

- We are familiar with Σ as the usually-defined alphabet.
- Q is a finite set of states. It cannot be empty because it contains q_0.
- q_0 is a designated initial state. ($q_0 \in Q$)
- F is a subset of Q, and is the set of designated final states (aka accept states). ($F \subseteq Q$) F may be empty, because it is simply defined as a subset.
- δ is a total function (it is always defined, over the entire domain) called the transition function. We define δ as $\delta : (Q \times \Sigma) \to Q$: for every pair (state, symbol), δ maps this pair to some state in Q.

Notation: If $\delta(q, \mathtt{a}) = p$, where $q, p \in Q$ and $\mathtt{a} \in \Sigma$, then we write:

$$[q, \mathtt{a}, p] \in \delta,$$

which is read "in current state q, on symbol $\mathtt{a}$, transition to next state p."

Concept: The *configuration* (snapshot) of a computation is a snapshot of the history of computation that is required and sufficient to continue.

The **configuration** of finite automaton M is a pair (q, w) where $q \in Q$ and $w \in \Sigma^*$. Here, q is the current state and w is unprocessed input (meaning, under the head and to the right).

A DFA starts in the initial configuration, of (q_0, w_0) for $q_0 \in Q, w \in \Sigma^*$. Then, recursively from the current configuration (q, w), the next configuration $(p, x), p \in Q, x \in \Sigma^*$ is calculated as follows:

The configuration is **terminal** if it is of the form $(t, \lambda), t \in Q$.

It is **accepting** if $t \in F$, and **rejecting** if $t \notin F$.

$L(M)$, the set of strings defined by M is the set of exactly those strings that M accepts; that is, that take M from the initial configuration to an accepting one.

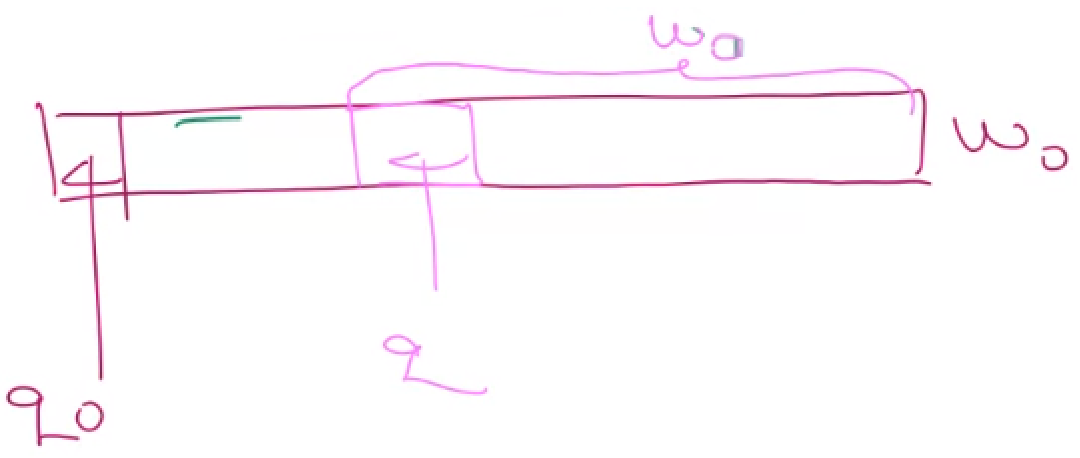

Figure 2: Illustration of the calculation of the next configuration

Example:

Notation (1):

- $M = (Q, \Sigma, \delta, q, F)$
- $\Sigma = \{\texttt{a}, \texttt{b}\}$
- $G = \{q, p\}$
- $F = \{p\}$
- $\delta : \cdots$
  - $[q, \texttt{a}, p]$
  - $[q, \texttt{b}, p]$
  - $[p, \texttt{a}, q]$
  - $[p, \texttt{b}, q]$

State transition diagrams

Notation (2) (state transition diagram):

A **state transition diagram** is a directed graph, with states as the nodes and transitions (the δ tuples) as the edges. The initial state has a funnel into it and the accept state has a circle around it.

We can also write the transition function as a table:

	a	b
q	p	p
p	q	q

We can see that the above automaton accepts strings of odd length.

The question, when trying to determine how an automaton works, is "which strings take M to this/these (accept) state(s)?"

Theorem: Let L be a language accepted by a deterministic finite automaton:

$$M = (Q, \Sigma, \delta, q_0, F)$$

In other words, $L = L(M)$.

Then, its **complement** $\overline{L}$ is accepted by:

$$M_1 = (Q, \Sigma, \delta, q_0, (Q \backslash F)).$$

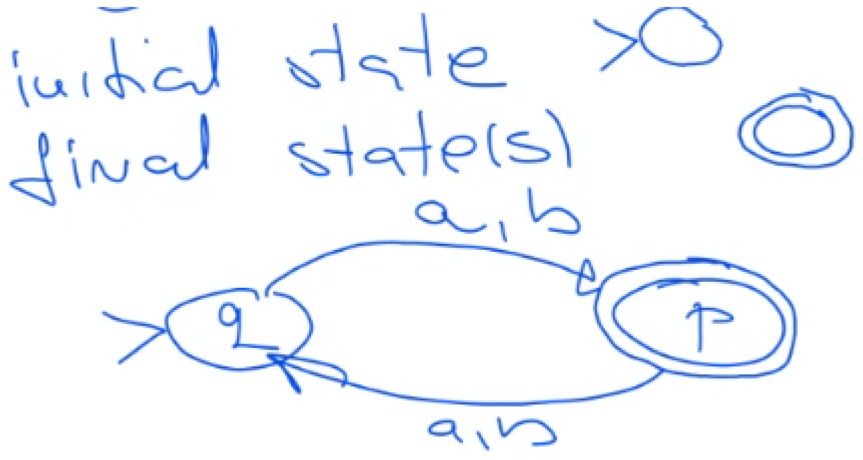

Figure 3: State diagram for above-given DFA, M

Simply by inverting the accept and reject states we can achieve the previously difficult notion of complement.

Example: Divisibility by 3

We'll need three states, to keep track of what remainder we have modulo 3.

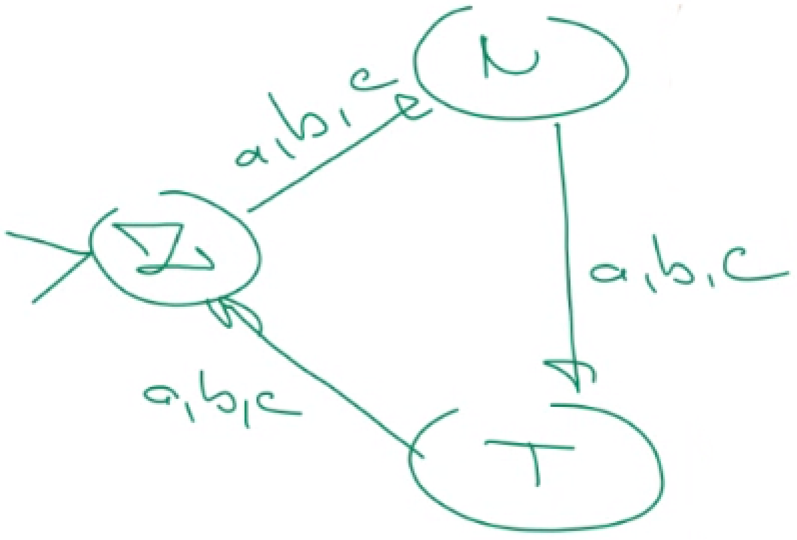

Figure 4: State diagram for divisibility by 3

If we want the strings of length divisible by 3, we take $F = \{Z\}$; if we want the even remainder, we take $F = \{Z, T\}$; if we want those not divisible, we take $F = \{N, T\}$.

Example: Let $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$. What is the DFA that accepts exactly those strings that begin with substring bbb?

Example: Contains substring bb.

For a string w, define n_a as the number of a's in w, define n_b as the number of b's in w, and define n_c as the number of c's in w.

For $w =$ cabacbacbbca we have $n_a = 3, n_b = 4, n_c = 5$.

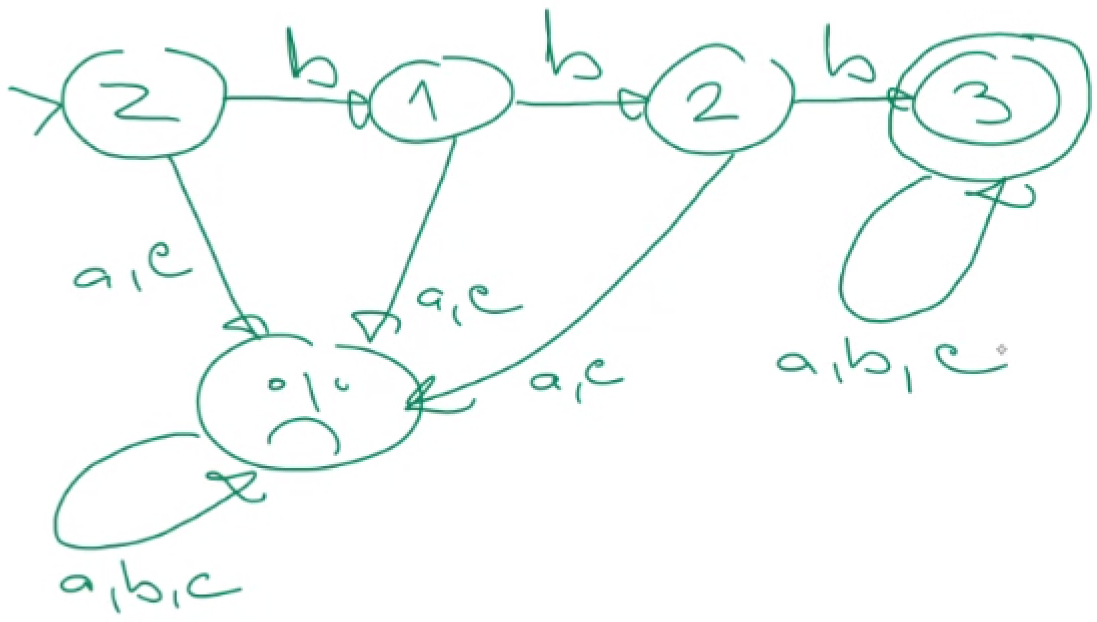

Figure 5: State diagram for DFA accepting strings beginning with `bbb`

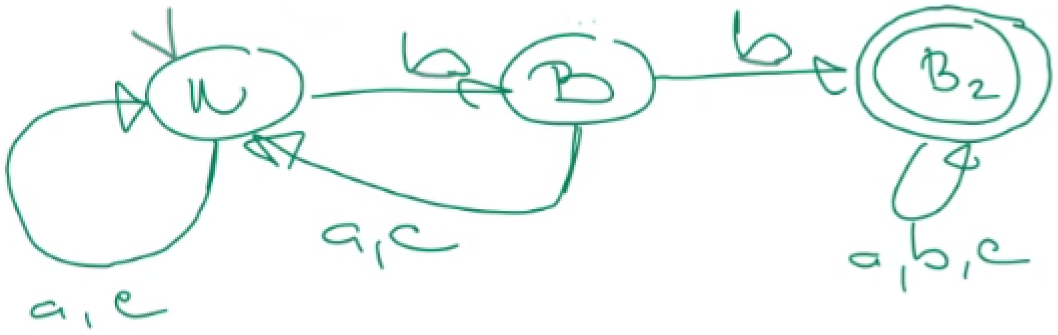

Figure 6: State diagram for DFA accepting strings containing `bb`

Construct a DFA to accept the set of strings for which:

$$2n_a + n_b - n_c + 3 = \alpha$$

gives an odd remainder after division by 5.

We will need 5 states, because there are 5 possible remainders.

Each `b` contributes 1 to α, each `c` subtracts one from α, each `a` contributes 2 to α. The initial state is $\textcircled{3}$, because if we consider λ as an input, we have $n_a = n_b = n_c = 0$ and then $\alpha = 3$. The final states will be $\textcircled{1}$ and $\textcircled{3}$ because they are the odd remainders.

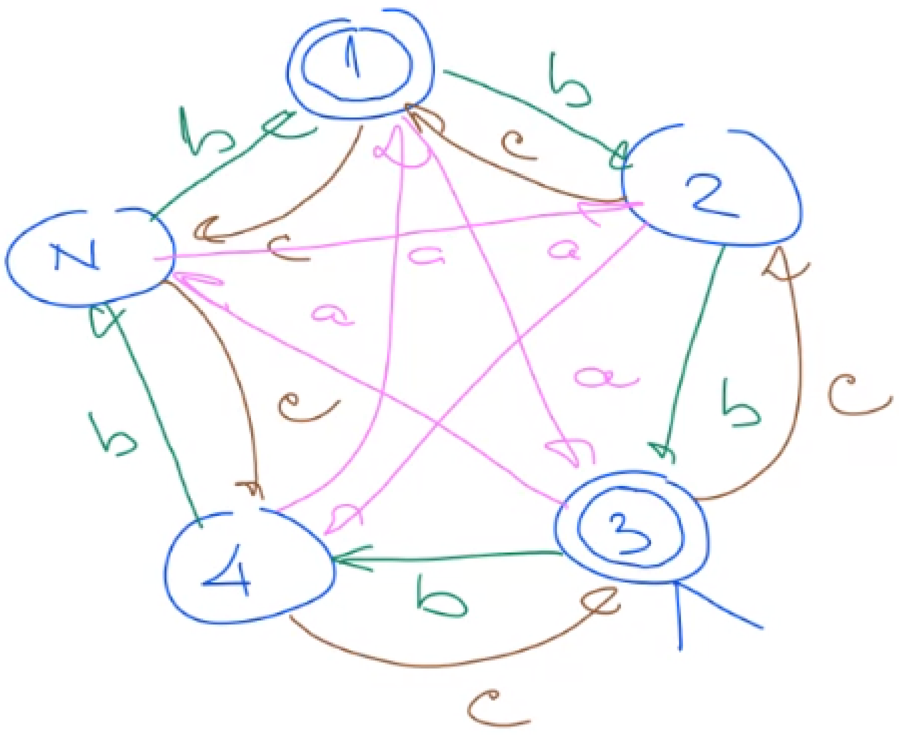

Figure 7: State diagram for division of α by 5 yielding an odd remainder

Theorem: Let L_1 be a language accepted by a finite automaton:

$$M_1 = (Q_1, \Sigma, \delta, q_1, F_1)$$

and let L_2 be a language accepted by a finite automaton

$$M_2 = (Q_2, \Sigma, \delta, q_2, F_2).$$

Then their **intersection** $L_1 \cap L_2$ is accepted by:

$$M = ((Q_1 \times Q_2), \Sigma, \delta, (q_1, q_2), (F_1 \times F_2))$$

where δ contains tuple $[(p, q), a, (s, t)]$ exactly when

$$[p, a, s] \in \delta_1$$
$$[q, a, t] \in \delta_2$$

Idea: Have a composite state, which is a pair of states, that simulates both machines.

Example: $\Sigma = \{\texttt{a},\texttt{b},\texttt{c}\}$, and we want an even number of `b`'s and an odd number of `c`'s.

We need to count the number of `b`'s, and the number of `c`'s, using the same mod-2 arithmetic for both (as we only care about parity). We'll have 4 total states:

even `b`, even `c`	odd `b`, even `c`
even `b`, odd `c`	odd `b`, odd `c`

Somewhat trivially we can take care of `a` by having our outomaton stay in the same state.

A `b` will flip our state from even `b` to odd `b` or vice versa.

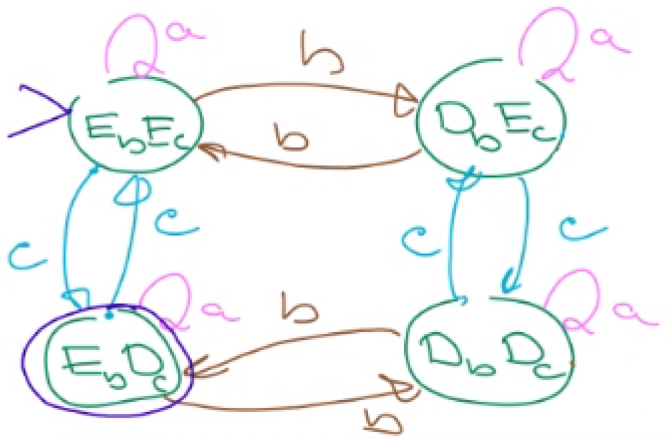

Figure 8: Composite state example

Example: Set of exactly those strings that *end* with `bbb`.

Idea: "Remember" last three symbols. If they are `bbb`, then accept; otherwise, reject.

Issue: There are 27 of them ($3 \cdot 3 \cdot 3$). There are also more states to reach this.

Example: Set of exactly those strings that contain `babbabbba`:

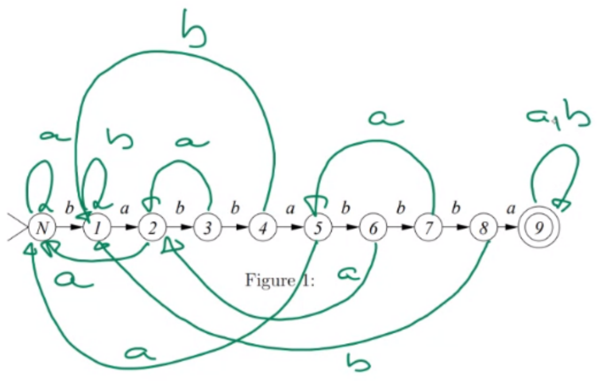

Figure 9: Set of exactly those strings that contain `babbabbba`

Tomorrow: NFAs (nondeterministic finite automatons)